NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

THE MERITS OF THE CONTINUED INSTRUCTION OF ADA AS A FIRST LANGUAGE AT THE NAVAL POSTGRADUATE SCHOOL

by

Thomas C. Gomez

September, 1994

Thesis Advisor: David A.Gaitros

REPORT DOCUMENTATION PAGE			Form Approved OMB No. 0704
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instruction, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington DC 20503.			
1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE September 1994	3. REPORT TYPE AND DATES COVERED Master's Thesis	
4. TITLE AND SUBTITLE: THE MERITS OF THE CONTINUED INSTRUCTION OF ADA AS A FIRST LANGUAGE AT THE NAVAL POSTGRADUATE SCHOOL			5. FUNDING NUMBERS
6. AUTHOR(S) Thomas C Gomez			
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey CA 93943-5000			8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)			10. SPONSORING/MONITORING AGENCY REPORT NUMBER
11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.			
12a. DISTRIBUTION/AVAILABILITY STATEMENT Approved for public release; distribution unlimited			12b. DISTRIBUTION CODE A

13. ABSTRACT *(maximum 200 words)*

This thesis addresses the issue of the continued instruction of structured programming in general and Ada in particular as the first programming language at the Naval Postgraduate School. The catch-22 of industry's dedication to C++ and the Department of Defense's support of Ada makes the choice of the proper language at a military graduate school difficult. The change to the present curriculum provides an opportunity to collect valuable data upon which to base this decision.

The approach was to identify the relative strengths and weaknesses of Ada and C++ as they pertain to first quarter non-computer science undergraduates and meeting the needs of Department of Defense Directives. Additionally, a set of programming projects to be solved by students in both language was generated. Analysis of the students' work will provide another set of data points to make an informed decision.

Based on its reliability, standardization and its Department of Defense support, we conclude that Ada9X offers significant advantages over C++ and should be selected as the first programming language. Ada9X offers both the object oriented paradigm and is in line with the Department of Defense's commitment to Ada for non-COTS applications.

14. SUBJECT TERMS Ada, C++, structured programming, object oriented programming			15. NUMBER OF PAGES 46
			16. PRICE CODE
17. SECURITY CLASSIFICATION OF REPORT Unclassified	18. SECURITY CLASSIFICATION OF THIS PAGE Unclassified	19. SECURITY CLASSIFICATION OF ABSTRACT Unclassified	20. LIMITATION OF ABSTRACT UL

NSN 7540-01-280-5500

Standard Form 298 (Rev 2-89)
Prescribed by ANSI Std. 239-18

The Merits of the Continued Instruction of Ada as a First Language at the Naval Postgraduate School

by

Thomas C Gomez
Lieutenant, United States Navy
B S , United States Naval Academy, 1988

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
September 1994

Author: [illegible]
Thomas C. Gomez

Approved by: [illegible]
David A. Gaitros, Thesis Advisor

[illegible]
Frank Kelbe, Second Reader

[illegible]
Ted Lewis, Chairman
Department of Computer Science

ABSTRACT

This thesis addresses the issue of the continued instruction of structured programming in general, and Ada in particular as the first programming language at the Naval Postgraduate School. The catch-22 of industry's dedication to C++ and the Department of Defense's support of Ada makes the choice of the proper language at a military graduate school difficult. The change to the present curriculum provides an opportunity to collect valuable data upon which to base this decision.

The approach was to identify the relative strengths and weaknesses of Ada and C++ as they pertain to first quarter non-computer science undergraduates and meeting the needs of Department of Defense directives. Additionally, a set of programming projects to be solved by students in both languages was generated. Analysis of the students' work will provide another set of data points to make an informed decision.

Based on its readability, standardization and its Department of Defense support, we conclude that Ada9X offers significant advantages over C++ and should be selected as the first programming language. Ada9X offers both the object oriented paradigm and is in line with the Department of Defense's commitment to Ada for non-COTS applications.

TABLE OF CONTENTS

I. INTRODUCTION

The Computer Science Department at the Naval Postgraduate School is planning to alter the current curriculum. This change replaces CS2970 Structured Programming with Ada with CS2971 Introduction to Programming with C++ and CS2972 Introduction to Programming with Ada. This change takes effect in the fall quarter of Academic Year 1995, during which the student must decide which of these two courses he wishes to take prior to the start of the first quarter. Such a decision affords an excellent opportunity to evaluate the relative strengths of each language, especially in satisfying the goals of the outgoing CS2970 course. This thesis will address the value of the continued instruction of the structured programming paradigm using Ada in a military graduate education program.

A. BACKGROUND

According to the Naval Postgraduate School Catalog for Academic Year 1994, the purpose of CS2970 is twofold:

1. It serves as an introduction to problem solving, and
2. It introduces a programming language.

Exposure to C++ in the current curriculum, if desired, comes in CS3700, Advanced Programming in C++. An elective, CS3700 has as its prerequisites completion of CS2970 and CS3300 Data Structures. Currently, Ada is the only language used in CS3300, but the course will become language independent when the new curriculum takes effect.

Within the Computer Science Department there are faculty members and students adamantly opposed to the change, while at the same time there are members who claim such a change is long overdue. Those in favor of retaining CS2970 and Ada recognize its power and value as an easily maintained high level language useful in software engineering applications as well as a general purpose problem solving. Those in favor of replacing CS2970 feel the need to begin instruction of an object oriented design methodology is necessary in order for the curriculum to keep pace with the dynamic computing world. Some even view Ada as an archaic language forced upon them by the government in effort to keep the language alive.

B. RESEARCH QUESTIONS

This thesis will address the following questions which are broken down into four catagories:

1. What is being changed?
   a. What is structured programming?
   b. What is object oriented programming?
2. Why is it being changed?
   a. Is C++ a fad language?
   b. Will the new curriculum satisfy the same goals of CS2970?
   c. Can students make an informed decision when choosing between CS2971 and CS2972?
   d. Why is Ada important to the Department of Defense?

3. What are the respective strengths and weaknesses of Ada and C++?

   a. What capabilities does OOP offer?

   b. Do Ada or Ada 9X offer similar capabilities?

4. Are the advantages associated with an object oriented paradigm sufficient to outweigh the advantages of a structured programming paradigm to better understand problem solving techniques?

C. METHODOLOGY

This thesis will report the value of the continued education of structured programming paradigms in the military graduate education program. Given the military's commitment and requirement to keep costs down, it will look at the value of both structured programming and object oriented programming and their impacts on the requirements definition, design, coding, testing and maintenance phases of a program's lifecycle. This will be accomplished by devising five problems to be solved by students once the new curriculum is in place. By tracking important statistics during the various software design phases one can make an informed decision regarding the satisfiability of goals each language and their associated paradigms offer. As such, a recommendation as to an improved curriculum can be made once some data can be collected and evaluated.

D. FUTURE WORK

This thesis topic offers considerable future work. Such work includes, but is not limited to:

1. Among them is the implementation of the proposed programming problems and the evaluation of the collected data.
2. Automation of the data collection.
3. Continuation of the study during the Data Structures course. This would require the design of additional problems.
4. Proposal of improvements to the curriculum based on the analysis of the data collected in 1 and 3 above.
5. Development of benchmark program solutions by more experienced programmers to the proposed problems for comparison to student solutions.

The proposed future work topics can be undertaken either singlularly or as a combination of two or more. Collection and analysis of the data over several quarters would best represent any possible problems with the new curriculum, especially as new students come into the program with presuamably more computing experience in undergraduate programs. This may or may not be the case. A student profile outlining prior programming experiences would assist in the evaluation.

II. EVOLUTION OF PROGRAMMING LANGUAGES

Before discussing why such a change to the curriculum is being made, it is important to first understand what is being changed. The two paradigms are arguably equally important and to a certain degree related. This becomes especially true with the evolution of new languages such Ada 9X which has already been authorized to be used in its beta version. (Paige, 1994) As of the writing of this thesis, there is no officially released version of Ada9X.

A. ORIGIN OF STRUCTURED PROGRAMMING

The ability to compute was recognized early in the development of computer science. As such, there was a growing demand for software to handle problems that ranged from the mundane to critical. The quantity of software and its complexity, of course, must be tempered to the times and the state of computing at the time. In the early 1960s the industry began experiencing some difficulties. Scheduled delivery of software was overdue, budgets for procuring software was exceeded and frequently, the software was unreliable. As the need for more complex software rose, it became evident that programming was a much more involved process. This realization was the birth of the structured programming paradigm. (Deitel and Deitel, 1994)

The structured programming paradigm is a disciplined approach to writing programs that results in clearer programs than those from an unstructured approach. The complexity of programs revolved around the transfer of control. (Deitel and Deitel, 1994)

Transfer of control occurs when the order of execution of steps in a program is no longer sequential. The shift from sequential execution was accomplished with the "goto" statement which allowed the programmer to specify transfer of control to virtually anywhere within the program. The goal of structured programming, among other things, was elimination of goto statements. The structured programming approach resulted in clearer programs that were easier to read and debug, and in fact were more likely to be bug free. Additionally, structured programs became easier to modify and maintain than their unstructured counterparts. What could not be denied was the fact that more software designs were delivered on time, within budget and more correct. (Fastrack, 1990)

1. Evolution of Ada

Ada evolved out of a realization in the Department of Defense that a standard programming language for use in military applications was needed. The number of computer languages and dialects in use (400 - 1500), the evolving hardware technology and the lack of software tools yielded a difficult to manage software crisis. The code was not transportable across different platforms and software systems were increasing in size and complexity. A Department of Defense study conducted between 1973 and 1974 revealed that DOD was spending $3 billion each year for software and that maintenance costs exceeded the cost of software development. (Cohen)

In January, 1975, the High Order Language Working Group (HOLWG) was chartered to unify Army, Navy, and Air Force efforts to develop a common language in lieu of an individual service language. The goals of HOLWG were to establish a standard military language and develop a common set of language oriented tools. An additional benefit would be the feasibility of developing an integrated, language oriented software development and maintenance environment that would integrate the software tools and new library facilities. The HOLWG determined that no single existing language would meet their requirements and concluded that Pascal, ALGOL 68, and PL/1 would be the basis for a new language. The language would satisfy the following criteria: reliability, maintainability, and efficiency. (Amoroso and Ingargiola, 1985) These requirements were realized in Ada83.

The nature of military systems requires a software package that is both reliable and efficient. The software must perform the same every time and must run quickly, especially in combat systems. The inelegant handling of an exception and resulting program crash could mean serious equipment damage, disruption of command and control systems at crucial times, and loss of life. Maintainability of software is necessary given the lifespan of military systems. In order to keep maintenance costs down, the software must be released with no bugs. (Fastrack)

The Department of Defense initiated the Ada project to address the increasing costs associated with maintaining its software. It was willing to increase the cost of developing software if it could ensure decreased maintenance costs. It recognized the need to shift from a brute force hard coding of the problem to a software engineering approach. Software engineering is the establishment and use of sound engineering principles in order to obtain economical software that is reliable and works efficiently on real machines. (Fastrack, 1990) The Department of Defense and other government agencies such as NASA remain committed to Ada.

B. ORIGIN OF OBJECT ORIENTED PROGRAMMING

The explosion of software needs continued. The demands were such that designing software quickly, economically and correctly was as key a factor as the increased power and complexity of the software. A new programming paradigm evolved to meet these growing challenges: object oriented programming.

As the name implies, object oriented programs utilize objects. An object is a reusable software component which models a real world entity. The capability to reuse objects intuitively yields faster software design implementation. Objects have a state defined by their attributes and they interact with other objects via their methods. Each usable object has a set of attributes and methods. The perception is that object oriented design provides significant improvements to software designed using a structured approach. This improvement is significant in the implementation phase of software development. The realization of these perceived improvements in software development is through the concepts of inheritance, polymorphism, encapsulation and abstraction.

Inheritance is the most commonly referred to aspect of object oriented programming, and rightly so. Inheritance provides for software reusability. It does this by allowing a class to act as a parent of one or more new classes. The new class "inherits" the parent or base class attributes and methods relieving the programmer of writing new code to represent something that resembles something he has already written. He can then improve the child or derived class to meet the needs of the problem solution. With this in mind, a well designed parent class must be specific enough to accurately represent the real world object, but general enough to be useful as a base class. The number of base class attributes and methods should be sufficient such that they are useful to any derived classes. Inheritance comes in two flavors: single and multiple inheritance. A derived class that has one base class demonstrates single inheritance. As the name implies, multiple inheritance describes the origin of a derived class with two or more base classes.

In order to realize the benefits afforded by inheritance the concept of polymorphism is key. Polymorphism allows objects related through inheritance to react as required when a method that is common to more than one class is invoked. The derived class inherits all of the base class' methods. These methods will have the same name. The need to distinguish between the two is obvious.

Encapsulation provides for information hiding. An object encapsulates the attributes and methods of a real world entity. Objects can interact via well defined interfaces, but have no need to know how the other is implemented. The ability to hide the details of a programs implementation is crucial to software engineering.

Abstraction can be described as a means of seeing the forest through the trees. It provides for the hiding of certain details to allow more general thinking and to allow separation of the problems into independent subproblems.

The claim of supporters of object oriented design is that object oriented programming allows for simpler design because it more closely matches the way people think, namely in the sense of objects interacting amongst themselves in the same way objects (i.e. people and things) interact in the real world.

C. HYBRID LANGUAGES

Hybrid languages combine the aspects of both structured programming and object oriented programming. Most notable among hybrid languages is C++, an offshoot of C, a structured language. C++ is widely used by many of today's programmers.

1. Evolution of C++

C++ is the premier programming language in the computing world today. C programmers recognize the "++" to represent the increment operator. Thus, C++ is an increment of C. While it is true that C++ holds the C language as a subset, it is much more than an increment of C. C++ is an expansion of C with Classes that incorporates the addition of operator overloading, virtual functions and references.

Bjarne Stroustrup developed C++ so that he and his friends "would not have to program in assembler, C, or various modern high level languages." (Stroustrup, 1991) It became available, in a preliminary version, outside of this group in July, 1983. After it was released, it continued to evolve based on the suggestions of its users as additional nice to have features were developed to overcome problems and other shortcomings. As a result, there are multiple versions of C++ compilers in existence, each supporting different capabilities. There is an effort underway to develop an ANSI standard version of the language. As a hybrid language, C++ supports both structured programming and object oriented programming concepts. There are many software engineering tools available to assist software designers in formulating sound code.

III. ADA OR C++

Both Ada and C++ originated in the early 1980s and were derived from pre-existing languages. These two high level languages have their ardent supporters and opponents; usually a supporter of one language is an opponent of the other. Ada's support is found in software engineering, Department of Defense applications, and has a surprisingly large following in Europe. C++ has support throughout computer science. Each language is suitable for solving the same types of problems. With respect to choosing between an object oriented language such as as C++ and the continued instruction of structured programming in general and Ada in particular in the military graduate education program, the question that must be answered is: What are the strengths and weaknesses of each language? An informed decision can be made based on the answers to this question.

While Ada and C++ are both powerful languages capable of providing adequate solutions to various programming problems, they are still separate languages As such they each have their respective strengths and weaknesses, some of which are common to both. In this chapter, a comparison of the two languages will be made. The means with which each language implements certain features will be briefly discussed.

A. COMMON FEATURES

Ada and C++ are both powerful languages. Despite the disparity among their respective supporters, Ada and C++ share some common features. Among them are modularity, information hiding and encapsulation. These features are critical to the success of today's programs.

Program modularity is an important capability. Given the size and complexity of program solutions, the ability to support a modular design is critical. A modular design makes it possible to pursue a "divide and conquer" approach in which the large problem is divided into less complex subproblems. With especially large programming solutions, numerous programmers are used to code the solution. Modularity facilitates the division of labor amongst the programmers as each can be assigned responsibility for a number of modules.

Encapsulation is discussed in Chapter II. Briefly, however, encapsulation groups together related declarations and subprograms. These can then be stored in a library for future use.

A fundamental principle of good software design is information hiding. Information hiding is a crucial capability and comes as a result of program modularity and encapsulation. At the same time, the success of modularity hinges on information hiding. Through information hiding, the implementation details of a module are not available outside that module. The various modules communicate via a well defined interface, but the specifics of how this is accomplished remains within the respective module.

1. Implementation in C++

In C++, modules are called functions and classes. These are considered to be program structuring units that can be used to build objects. An object is an instance of a class which can be used to model an entity that can be expressed in terms of attributes and smaller modules called methods. The methods are written as functions which may or may not take a parameter and will return a value (which could be null). These functions written by the programmer can be combined with the pre-packaged functions available in existing class libraries to yield the object.. A method is invoked via a message. Therefore, a message can be likened to a function call. (Deitel and Deitel) To satisfy the "divide and conquer" approach, a programmer can be assigned responsibility for coding an object and its associated functions.

The encapsulation of data takes place in the object. This is a key premise of object oriented design. An object holds the attributes and methods of an entity and can be stored for later use, supporting code reusability.

Objects are said to have the property of information hiding. It is not necessary for each object to access another object's member functions. Interaction with other objects is through the well defined interfaces between objects. There is no need for an object to know how another is implemented. (Stroustrup)

2. Implementation in Ada

In Ada, modularity begins as an abstract data type. An abstract data type is a data structure and a collection of operations on the data structure which provides for generality and modularity. The operations of an ADT are written as procedures and functions. Procedures and functions are the basic modules of an Ada program. Procedures and functions are sequences of statements identified by their names. Invoking the name of the procedure or function executes the steps in their bodies.

A procedure is a small program. The call to a procedure may include a list of parameters. These parameters may be values to be used by the procedure or variables which are to be changed by the procedure. A function differs from a procedure in that the parameters may be values to be used by the function. The function uses these values to compute another value which the function then returns to the program where it was called.

The concept of encapsulation in Ada is realized by the use of packages. A package provides a storage place for related declarations and program modules. These modules can then be stored in a library for reuse.

A package consists of two parts: its specification and its body. The specification of a package tells what is available in that package and provides the interface between the user and the package. The body of the package contains the details of how the available modules are implemented. Within the body of a package can be a private section. The private section will be hidden from view outside of the package, hiding the details of the implementation.

B. ADA SPECIFICS

The Department of Defense remains committed to Ada where non Commercial Off The Shelf (COTS) applications are available as DOD will inherit the lifecycle costs of maintaining that software. This section will discuss the relative strengths and weaknesses of Ada.

1. Ada Advantages

The first goal of Ada is program reliability and ease of program maintenance. (Intermetrics) The Ada project has made a significant step toward achieving this goal by making the language readable. The syntax closely resembles English. When reading the lines of code of an Ada program, it can be readily understood. The logic may require some additional inspection, but the individual steps are clear. This readability lends itself to increased reliability and eased maintenance costs. If a programmer can understand what he wrote when he reads it next year as easily as when it was originally written, it is more likely to be correct. A program which is readable is likely to be easier to modify later as the programmer spends less time toiling over countless lines of cryptic code. (Schoneberg) This is especially helpful in the environment of graduate education at the Naval Postgraduate School given the various programming backgrounds of its students.

The fact that Ada is a standardized language is a significant plus. The language and its environment do not have various versions and does not rely on the operating system as do other languages. (Schoneberg)

Ada code is said to be safe. Ada requires the programmer to declare all variables in the header prior to using them. The provision for exception handling demonstrates the forethought that went into the design of the language. These aspects again stress the importance a good design.

The impact of readability on program maintenance shifts both the effort and costs toward the beginning of program inception resulting in a more normal distribution. (Fastrack) More effort is spent in the design phase of the program lifecycle and less is spent in the coding and testing phases. The longer design time supports good naming conventions, well designed packages and error handling. These factors directly support readability and maintenance, code reuse and reliability.

2. Ada Disadvantages

Ada is not without its disadvantages. First, a concern among Ada supporters is the increasing size of Ada, especially with the approaching release of Ada9X. Ada is a complex language whose success is dependent on its users knowing how to utilize all of its features to yield good programs. (Fastrack) The addition of new features associated with Ada9X could conceivably scare off some supporters, citing that the language is now too complex to be useful. A similar situation can be seen in the downfall of IBM's PL/1. By requiring the use of built in libraries the user's ability to create datatypes was restricted. The solution for added capability was the addition of libraries to provide the users' with the widgets they required. The added complexity was sufficient to ensure the demise of PL/1.

Since Ada does not share the same popularity as other programming languages, there are fewer programmers keenly versed in the Ada language and its approach. This is a potentially serious problem early in the life of a language, and the insistence of DOD on

Ada implementations could require "home grown" programmers to fulfill its programming needs. Now that waivers to permit programming in languages other than Ada are being denied (Paige), Ada could realize increased popularity out of necessity.

The final shortcoming of Ada deals with the increased design time. First, a customer could become overly concerned when software progress appears slow, placing undue pressure on program managers. So long as both customers and program managers realize this front loading in the design phase is a necessary evil of Ada, this problem will prove minimal. (Fastrack) Also associated with the increased design phase is the fact that problems with package specifications and program logic may not be realized until much effort has been expended. The need for the developer to fully understand the customer's desires is critical. (Yourdon).

3. Conclusions

The Department of Defense commitment to Ada seemingly makes Ada a popular language out of necessity. This is not necessarily a bad thing. Ada83 proved to be a good language and with the additions Ada9X brings, it could be even more powerful and in line with the mainstream with respect to object orientation. The development of Ada is based on a significant research effort on the part of the HOLWG. As such, Ada is a well thought out language that is standardized. Its readability and ease of maintenance makes it an ideal language for students in military graduate education programs. Many students who leave such programs go on to follow-on tours, some of which include becoming program managers. Understanding the Ada language and its approach to software design could only benefit such programs.

C. C++ SPECIFICS

C++ enjoys the position of being one of the most popular programming languages in computing today. This section will discuss the relative strengths and weaknesses of C++.

1. C++ Advantages

C++ is a programmers language. This means that limitations associated with other languages are not as restricting in C++, it was ,after all, designed by a programmer for use by himself and his friends. C++ provides many shortcuts and abbreviated statements. Variables may be declared as they are needed and operators such as the increment operator make common program lines even more simple. (Stroustrup) The ability to code on the fly can be a true plus.

The popularity of C++ equates to a plethora of programmers. Former C programmers can make an easier transition to C++, and there is no shortage of newcomers eager to make their mark in the C++ world. This could spell a potential increase in competition for contracts resulting in more competitive bidding.

The popularity of C++ also means that much effort will be made to continually improve the language. It is at the forefront of mainstream computing and there are few languages that pose a threat to its continued popularity.

2. C++ Disadvantages

The most negative thing that can be said for about C++ is the fact that no standard exists. Although efforts are currently underway to publish a standard version, none is close at this writing. (Schoneberg) Without a standard, code may not be readily reusable or platform independent. This is a severe limitation when there are more and more published libraries available for use.

Another drawback of C++ is the fact that it is difficult to read. One of its strengths yields a serious problem. Since C++ was written by a programmer for his own use a person less experienced or inexperienced in programming C++ would have a difficult time reading the cryptic lines of code. This also makes it more difficult for all but the most experienced programmers when it comes to maintaining or modifying the code later. (Schoneberg)

The fact that the Department of Defense will no longer be permitting waivers to allow languages other than Ada will have a direct impact on the design of new military systems. While those systems already in place with non Ada code will require C++ programmers, these systems will not last forever and nobody knows what the future holds for the language.

3. Conclusions

Much can be said for C++. It has come a long way from being used by a few colleagues to where it stands today. Given that C++ has been in use for ten years or so clearly indicates that C++ is not a fad language. The fact that millions of people program in C++ can attest to that. It has paved the way for mainstream object oriented design.

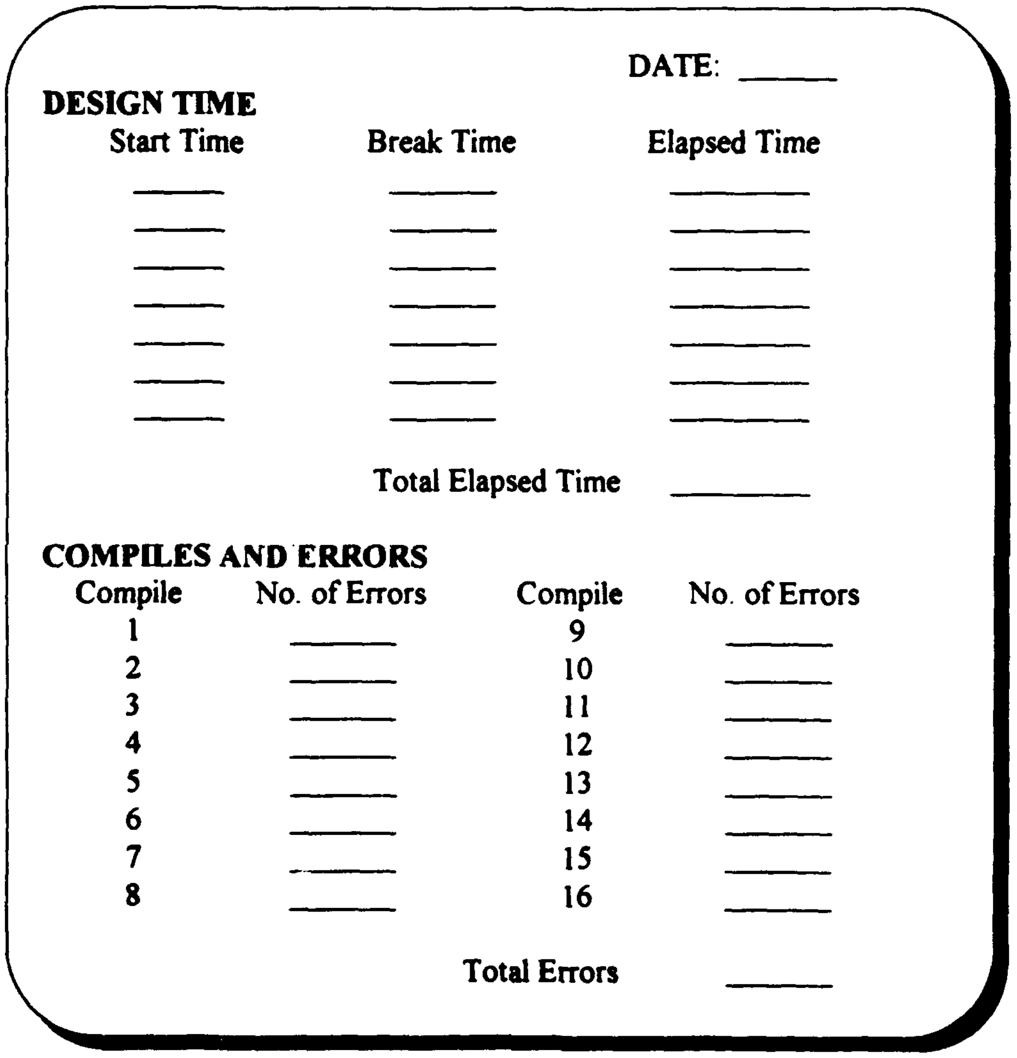
The question of the applicability of C++ to military graduate education remains. With the Department of Defense's commitment to Ada, the not too distant future of C++ in DOD systems is questionable. The extensive use of C++ in such projects as NPSNET require its instruction, but C++ should not be a required language for first quarter computer science students.

IV. PROGRAMMING PROJECTS

In order to obtain data for analysis a series of programming projects to be coded by first quarter students must be designed. The data will be derived from the code written by the students in their respective languages. The projects that follow closely parallel those required prior to the curriculum change. They must be of similar complexity to those programming assignments required in CS2970 in order to satisfy the same requirements and gain the necessary programming skills. The goal is to challenge the more exp^e·ienced students, but not overwhelm those who are less inexperienced.

The analysis to be performed on these programs will be both objective and subjective. The students will be required to maintain records of the following: required design time, number of compile attempts, and number of errors per compile. Diligent use of a form similar to that in Figure 1 will assist the collection of this data. In addition to the data compiled by the students, the number of modules, the number of lines of code per module, the total size of code, the total number of lines of code, execution time can be readily determined. A subjective evaluation of the program design will obviously require code inspection.

Such an experiment is not original. A similar experiment is addressed by Henry and Humphrey. This experiment analyzes the maintainability of code by a similar group of student programmers. The results of the data realized by the implementation of this thesis will be more applicable to determining the value of the continued education of Ada in the military graduate education program in that the majority of the students will have little programming experience. What will prove critical to both the Ada and C++ courses will be an in depth instruction concerning the importance of software design to yield superior programs.



DATE: ______

DESIGN TIME

Start Time	Break Time	Elapsed Time
______	______	______
______	______	______
______	______	______
______	______	______
______	______	______
______	______	______
______	______	______

Total Elapsed Time ______

COMPILES AND ERRORS

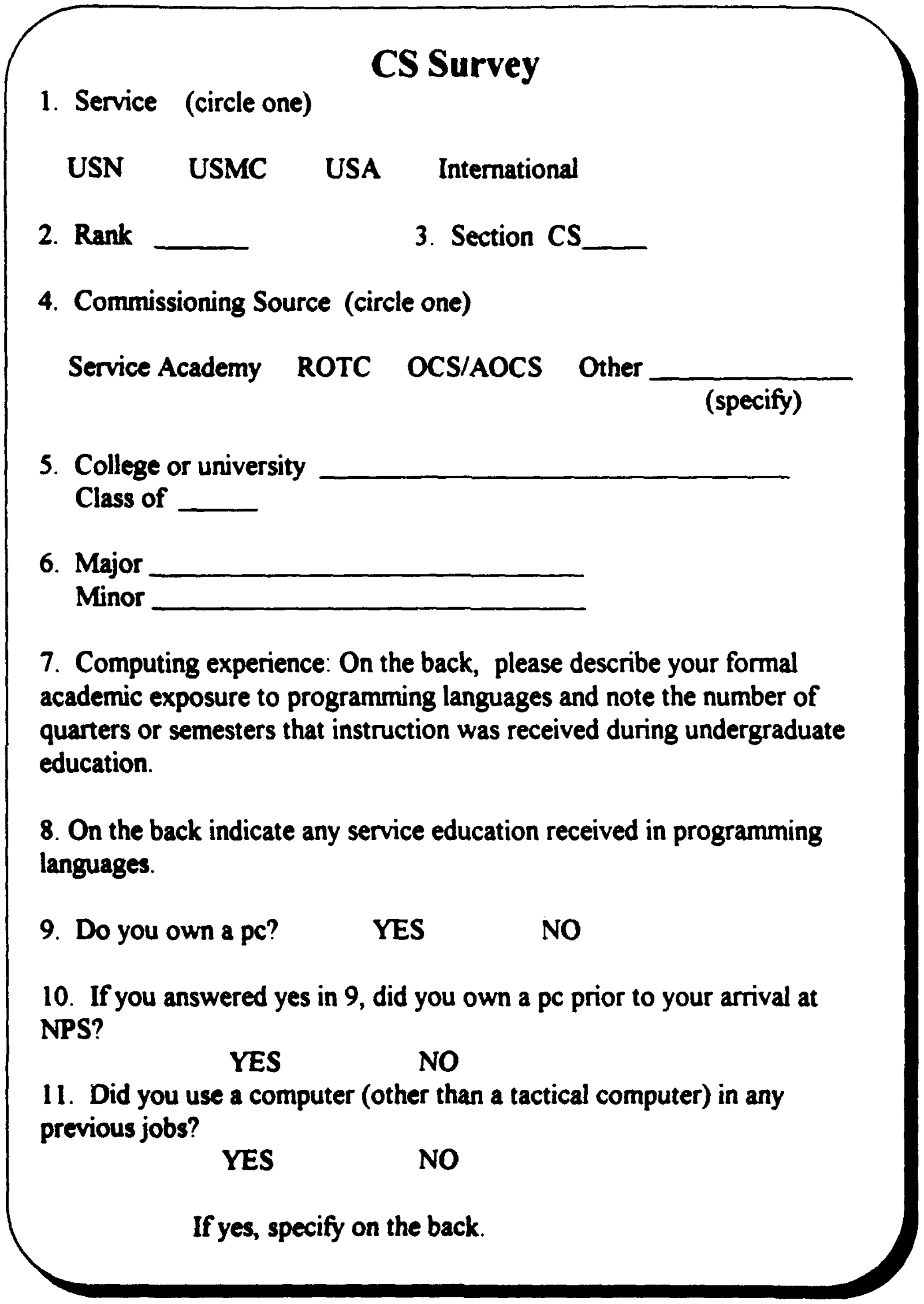
Compile	No. of Errors	Compile	No. of Errors
1	______	9	______
2	______	10	______
3	______	11	______
4	______	12	______
5	______	13	______
6	______	14	______
7	______	15	______
8	______	16	______

Total Errors ______

Figure 1

In order to satisfy the same requirements as CS2970 as the first programming language five programs will be written by the students. A sixth assignment could be included that would require the students to modify a program provided to them in order to gain data to address the issue of modifiability. Project one will expose the students to the programming language by requiring the performance of simple calculations and the utilization of the output functions. Project two will be introduce the students to classes, but may be optional. Projects three through five will stress the use of a modular design, adequate documentation, and the absence of global variables. Project three will require the manipulation of a text file and introduce the use of procedures. Project four will require the use of functions and procedures, parameter passing and exception handling. Project five will expose the students to recursive routines. Projects two through five resemble projects from previous quarters from both CS2970 and CS3700.

The availability of program solutions from previous students is a hazard when using projects from previous quarters which may affect the successful completion of the projects. This must be a consideration during analysis. Also of interest will be demographic background of the students. Completion of a survey form similar to that in Figure 2 will assist in minimizing the impact of programming outliers whose previous computer experience may skew the results of the analysis of code.



CS Survey

1. Service (circle one)

   USN USMC USA International

2. Rank ______ 3. Section CS____

4. Commissioning Source (circle one)

   Service Academy ROTC OCS/AOCS Other ____________ (specify)

5. College or university ______________________________
   Class of _____

6. Major ____________________________
   Minor ____________________________

7. Computing experience: On the back, please describe your formal academic exposure to programming languages and note the number of quarters or semesters that instruction was received during undergraduate education.

8. On the back indicate any service education received in programming languages.

9. Do you own a pc? YES NO

10. If you answered yes in 9, did you own a pc prior to your arrival at NPS?

    YES NO

11. Did you use a computer (other than a tactical computer) in any previous jobs?

    YES NO

If yes, specify on the back.

Figure 2

A. PROJECT ONE

Pythagorean Theorem

Problem Statement

The Pythagorean Theorem can be used to determine the length of a side of a right triangle given the lengths of the other two. A right triangle is a triangle in which one of the angles of the triangle is equal to 90 degrees. The side opposite of the right angle is known as the hypotenuse. The length of the hypotenuse, as defined by the Pythagorean Theorem is equal to the square root of the sum of the squares of the two sides.

Write an Ada/C++ program that computes the length of the hypotenuse of a right triangle given the lengths of the other sides and gives the area of this triangle in square inches and square feet.

Implementation Details

1. Your program should query the user for the lengths of two sides of a triangle in inches.

2. The Pythagorean Theorem states $X^2 = Y^2 + Z^2$ where X is the length of the hypotenuse and Y and Z are the lengths of the other sides.

3. The area of a triangle is given by the formula: Area = 1/2 (b * h) where b is the length of the base and h is the height of the triangle.

4. 1 square foot = 144 square inches.

Example Output

```
                Right Triangle
Length of side 1                 12 inches
Length of side 2                 8 inches
Length of hypotenuse             14.4 inches
Area of triangle (sq in)         48.0
Area of triangle (sq ft)         0.33
```

B. PROJECT TWO

Yagi Antenna

Problem Statement

The Yagi array antennas make use of parasitic elements to produce a unidirectional radiation pattern. The element connected to the feed line is the driven element. A director is generally shorter than the driven element and is located at the "front" of the antenna. A reflector is generally longer than the driven element and is located at the "back" of the antenna.

Write an Ada/C++ program that computes the wavelength in meters for the given frequency, and the length of the elements of a three element Yagi antenna in meters and feet.

Implementation Details

1. Your program should query the user for the frequency in correct units (MHz). The program will take the frequency and display the calculated values. The user will then be prompted for a new frequency. This will continue until the user enters "0" for the frequency.

2. The antenna must be implemented as a class.

3. The output should include all elements shown in the example output, but may be formatted as you like.

4. Wavelength is given by the formula: $\lambda = c / f$.
where:
    λ = wavelength in meters
    c = speed of light in meters per second = $3 * 10^8$
    f = frequency in Hz

5. The length of the driven element (in ft) is given by: $L_{Driven} = 468 / f$.
The length of the director element (in ft) is given by: $L_{Director} = L_{Driven} * (1 - 0.05)$.
The length of the reflector element (in ft) is given by .$L_{Reflector} = L_{Driven} * (1 + 0.05)$.

6. 1 meter = 3.281 ft
1 MHz = 10^6 Hz

Example Output

```
Three Element Yagi Antenna
Frequency           28.6 MHz
Wavelength          10.49 m
Director Element     4.74 m    15.55 ft
Driven Element       4.99 m    16.36 ft
Reflector Element    5.24 m    17.18 ft
```

C. PROJECT THREE

String Utility

Problem Statement

One of the evils that faces you at the Naval Postgraduate School is the writing of papers. Since most writing assignments are given in terms of the number of words (such as a 1500 word paper), you would like to be able to count the words in your text file at various stages during the drafting of the paper.

Write an Ada/C++ program that will give you a ball-park idea of the number of words in your file. Ball-park because we will not concern ourselves with punctuation, words that spill over into the next line, etc. Additionally, your program should provide the following information:

number of characters,
number of words,
number of lines,
maximum/minimum word length, and
maximum/minimum line length.

Implementation Details

1. Write a program to read in a text file, display the file and output the statistics.

2. Concentrate on a clear, modular design. Do not use global variables.

3. A word is minimally defined as one or more printable characters. A blank line will count toward the number of lines, but will not used to determine the minimum word length or minimum line length. A space is considered to be a character.

4. Your program will be tested using the file PROJ2.TXT on the disk in the lab. Use a shorter text file of your own that will assist you in determining the correctness of your algorithm.

Example Output

If your text file is

```
Programming is not
a subject

about which I would like
to write a paper.
```

your output would be:

```
Programming is not
a subject

about which I would like
to write a paper.

Results of your file:
# Characters:        68
# Words:             14
# Lines:              5
Max word length:     11
Min word length:      1
Max line length:     18
Min line length:      9
```

D. PROJECT FOUR

SCUD Project

Problem Statement

A neighboring country is working on a missile project and you are its test area. Each morning CNN delivers you a disk which contains data of launches during one 24 hour period. Each data entry consists of the time of day of the launch and the range and elevation angle from the tracking station. Your boss wants a printout of all launches in a 24 hour period. He also needs to know the time and altitude of the first and last launch of the day as well as the time and altitude of the highest launch.

Write an Ada/C++ program to read the data file and provide this information.

Implementation Details

1. The file is organized so that the time, angle and range for each observation occupy one line. For example, the data line:

1 35 56 30.0 1130.5

represents a time of 01:35:56, an angle of 30 degrees and a range of 1130.5 meters.

2. Assume the file is in chronological order and the times are military time.

3. If the highest altitude was observed more than once in the same day, only report he latest launch information.

4. Use exception handlers to check for valid data. Do not process an obsevation with any of the following errors:
   - Incorrect type
   - Invalid time (hours > 23, minutes or seconds > 59)
   - Negative angle or > 90 degrees.
   - Negative range.

5. Altitude is given by the formula: Alt = c * sin(elevation angle) where c = range.

6. Concentrate on a clear, modular design. Do not use global variables. Make your procedures/functions communicate with parameters.

E. PROJECT FIVE

Raster Scan

Problem Statement

Imagine that you are writing some software for a radar system that will scan a digitized picture and identify the number of objects or entities on a single screen. A text file will be read that contains the "picture". Each line will contain either blanks or valid ASCII characters. The ASCII characters represent pixels of each entity. A single character is considered to be part of an entity if it is directly below, above or to either side of another character which is also part of the entity. Below is an example of two possible entities:

```
XXXXXXXXXXXXXXXXXXXXXXXXXX
 XXXXXX        XXXXXXXXXXXX
 XXXXX XAAAAX  XXXXXXXXXXX
 XXXXXX        XXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXX
```

Write an Ada/C++ program to determine the number of entities in a given text file.

Implementation Details

1. The input file will be no longer than 98 characters in length and no longer than 98 lines long. There may be blank lines, but each line will contain at least one blank or character.

2. Entities may be constructed of any combination on non-blank ASCII characters.

3. You will not be required to save the picture or print out the contents.

4. A character is considered to be part of an entity if it is directly below a character in the previous line, directly above a character in the following line, or to either side of a character in the same line.

5. Your program must read in the file and store the routine in a two dimensional array.

6. Your program must make use of recursive routines and follow the guidelines set down in this class for good modular design and documentation.

7. Your final run of this project must be done against the file ENTITY.DAT as provided in the laboratory on disk.

Example Output

```
There were 34 entities found in file ENTITY.DAT.
```

V. CONCLUSIONS AND RECOMMENDATIONS

This thesis addressed the strengths and weaknesses of both the structured and object oriented programming paradigms. It looked at some of the particular advantages and disadvantages of Ada, a structured programming language, and C++, an object oriented language. While this thesis merely lays the groundwork for basing a decision as to the continued instruction of Ada as the first programming language in military graduate education there are some conclusions that can be made prior to implementation of the projects outlined in Chapter IV. The impact of the object oriented features of Ada9X will affect the results regarding structured programming.

A. CONCLUSIONS

It is clear that the object oriented programming paradigm is here to stay; the popularity of C++ and other object oriented languages attests to that. While the mission of the Naval Postgraduate School, according to the school catalog, provides for "the advanced education of commissioned officers, and to provide such other technical and professional instruction, as may be prescribed to meet the technical needs of the Naval Service" and not those of the civilian computing culture, our officers will be dealing directly with these civilians in payback tours. As such, the computer science department must make object orientation a part of its curriculum if it hopes to maintain a curriculum in step with the current computing environment and provide its officers with the prevailing computing concepts. C++ and Ada9X will provide the vehicle for assuring NPS students are not left behind with respect to object orientation. This does not imply that the structured programming paradigm should be abandoned. On the contrary, the structured programming paradigm lives on in the object oriented paradigm. The fact that C++ and Ada9X were derived structured programming languages, namely C and Ada83, suggests that some knowledge of structured programming is necessary to program successfully utilizing an object oriented language.

The next stand out conclusion is the impact of the readability of Ada on both programmers and programs. A programming language that is readable has definite benefits for beginning programmers and experienced programmers alike. Making the transition from day to day thinking to thinking logically and putting those thoughts in the syntax of a programming language is difficult enough without having to utilize a cryptic language. Additionally, it has been documented that a program written in a readable language like Ada is easier to modify than a program written in a cryptic language. The effects of readability makes Ada an attractive language, especially for first quarter computer science students at the Naval Postgraduate School.

The fact that the Department of Defense continues to support the use of Ada despite its unpopularity and has seen fit to pursue an object oriented version speaks positively for Ada's future in government applications. With the shift toward a lifecycle cost approach for acquisition and DOD responsibility for non-COTS application maintenance, government is justifying front loading the costs for good designs associated with an Ada approach to software engineering provided maintenance costs are less. This necessitates the continued instruction of Ada as a requirement of all computer science students in military graduate education programs, not only because it would look bad if one of its own organizations did not support it, but because it makes sense. This, however, does not suggest that C++ should be neglected. The idea that a person with a masters degree in computer science could graduate today without exposure to C++ is unconscionable. The instruction of C++ further enhances an already exceptional program and bolsters the reputation of an education at the Naval Postgraduate School as being of the same caliber as one at a more prestigious university.

A course structure which results in the students ability to solve problems similar to those proposed in Chapter IV will satisfy the goals of CS2970. It will provide the opportunity to gain the basic problem solving skills required in the previous curriculum in addition to language specific orientation.

Giving the student the option of choosing a language prior to the first quarter is not advisable for several reasons. First, many students, especially those without previous programming experience, could not make an informed decision. Not all students are able to participate in a refresher quarter which could assist them in making such a decision. Additionally, it would prove to be a scheduling nightmare. Finally, it is reasonable that all computer science students share a common language, especially in the early stages of the curriculum.

B. RECOMMENDATIONS

It is recommended that Ada9X be used as the programming language for first quarter students at the Naval Postgraduate School for the reasons mentioned above. The results of the implementation of the experiment outlined in Chapter IV may indicate otherwise. It is also recommended that greater emphasis be placed on the art of software engineering as it applies to basic programming. This emphasis should come as instruction either prior to or in parallel with the instruction of Ada in the first quarter. While the needs of the respective services outweigh those of individual students, NPS in general and the Computer Science Department in particular should see to it that those students requiring a refresher, either the six week or full quarter, receive the opportunity to better prepare themselves for a challenging curriculum. A more arduous refresher period consisting of calculus and math logic, basic computer skills, and an introduction to software engineering would better serve non-computer science undergraduates in a period where the grades received do not count against one's graduate grade point average. Financial concerns could be overcome by limiting the refresher attendance to those students whose prior academic experience is in a field other than computer science.

Additionally, it is recommended that C++ become a required course for all computer science students to be taught in the third quarter. The third quarter is recommended so as not to overwhelm the students while they take Data Structures in the second quarter. By distributing the programming load one quarter the program will be both more enjoyable and academically rewarding to the students.

LIST OF REFERENCES

Amoroso, Serafino and Ingargiola, Giorgio, *Ada: An Introduction to Program Design and Coding*, Pitman Publishing, 1985.

Cohen, Norman H., *Ada as a Second Language*, McGraw Hill Book Company, 1986.

Deitel, H.M. and Deitel P.J., *C++ How to Program*, Prentice Hall, 1994.

Fastrack Training Inc., *Ada, A Management Perspective*, lecture notes, May 1990.

Henry, Sallie and Humphrey, Matthew, "Object Oriented vs. Procedural Programming Languages: Effectiveness in Program Maintenance," *Journal of Object Oriented Programming*, June 1993, p 41-49.

Intermetrics, Inc., *Programming Language Ada, Language and Standard Libraries*, Draft version 5.0, June 1994.

Naval Postgraduate School Catalog, Academic Year 1994.

Paige, Emmett, "Predictable Software: Order Out of Chaos," *Crosstalk*, June 1994, p 2 - 5.

Savitch, Walter J. and Petersen, Charles G., *Ada: An Introduction to the Art and Science of Programming*, Benjamin/Cummings Publishing Company, 1992.

Schonberg, Edmond, "Contrasts: Ada 9X and C++," *Crosstalk*, September 1992, p 12 - 16.

Stroustrup, Bjarne, *The C++ Programming Language*, Addison Wesley Publishing Company, 1991.

Yourdon, Edward, *Modern Structured Analysis*, Yourdon Press-Prentice Hall, 1989.

INITIAL DISTRIBUTION LIST

1.	Defense Technical Information Center Cameron Station Alexandria, Virginia 22304-6145	2
2.	Library, Code 052 Naval Postgraduate School Monterey, California 93943-5101	2
3.	Computer Technology, Code 32 Naval Postgraduate School Monterey, California 93943-5002	1
4.	Ted Lewis Chairman, Code CS Computer Science Department Naval Postgraduate School Monterey, California 93943-5118	1
5.	LTCOL David A. Gaitros, USAF, Code CS/Gi Department of Computer Science Naval Postgraduate School Monterey, California 93943-5002	1
6.	LCDR Frank Kelbe, USN, Code CS/Ke Department of Computer Science Naval Postgraduate School Monterey, California 93943-5002	1
7.	LT Thomas C. Gomez, USN 1289 Southfield Place Virginia Beach, Virginia 23452	1